**HTML**

<div id="container">

<canvas id="canvas"></canvas>

<div id="gridContainer">

<svg version="1.1"

id="grid"

viewBox="-50 -50 400 300"

fill="none"

stroke="none"

xmlns="http://www.w3.org/2000/svg"

xmlns:xlink="http://www.w3.org/1999/xlink">

<g transform="matrix(1, 0, 0, 1, 0, 0)">

<g stroke-width="0.25" stroke="#000000">

<line x1="0" y1="80" x2="0" y2="85"/>

<line x1="16" y1="80" x2="16" y2="85"/>

<line x1="32" y1="80" x2="32" y2="85"/>

<line x1="48" y1="80" x2="48" y2="85"/>

<line x1="64" y1="80" x2="64" y2="85"/>

<line x1="80" y1="80" x2="80" y2="85"/>

<line y1="0" x1="85" y2="0" x2="80" />

<line y1="16" x1="85" y2="16" x2="80" />

<line y1="32" x1="85" y2="32" x2="80" />

<line y1="48" x1="85" y2="48" x2="80" />

<line y1="64" x1="85" y2="64" x2="80" />

<line y1="80" x1="85" y2="80" x2="80" />

</g>

<g fill="#000000" font-size="5" font-family="monospace" text-anchor="middle">

<text x="0" y="90">-3</text>

<text x="16" y="90">-2</text>

<text x="32" y="90">-1</text>

<text x="48" y="90">0</text>

<text x="64" y="90">1</text>

<text x="80" y="90">2</text>

<text y="1" x="90">4</text>

<text y="17" x="90">3</text>

<text y="33" x="90">2</text>

<text y="49" x="90">1</text>

<text y="65" x="90">0</text>

<text y="81" x="90">-1</text>

</g>

</g>

</svg>

</div>

<div id="uiContainer">

<div id="ui">

<div id="wrap\_s">

<div>TEXTURE\_WRAP\_S: </div>

<div><input type="radio" id="wrap\_s0" name="wrap\_s" checked="true"/><label for="wrap\_s0">REPEAT</label></div>

<div><input type="radio" id="wrap\_s1" name="wrap\_s" /><label for="wrap\_s1">CLAMP\_TO\_EDGE</label></div>

<div><input type="radio" id="wrap\_s2" name="wrap\_s" /><label for="wrap\_s2">MIRRORED\_REPEAT</label></div>

</div>

<div id="wrap\_t">

<div><span>TEXTURE\_WRAP\_T: </span></div>

<div><input type="radio" id="wrap\_t0" name="wrap\_t" checked="true"/><label for="wrap\_t0">REPEAT</label></div>

<div><input type="radio" id="wrap\_t1" name="wrap\_t" /><label for="wrap\_t1">CLAMP\_TO\_EDGE</label></div>

<div><input type="radio" id="wrap\_t2" name="wrap\_t" /><label for="wrap\_t2">MIRRORED\_REPEAT</label></div>

</div>

</div>

</div>

</div>

<!-- vertex shader -->

<script id="vertex-shader-3d" type="x-shader/x-vertex">

attribute vec4 a\_position;

attribute vec2 a\_texcoord;

uniform mat4 u\_matrix;

varying vec2 v\_texcoord;

void main() {

// Multiply the position by the matrix.

gl\_Position = u\_matrix \* a\_position;

// Pass the texcoord to the fragment shader.

v\_texcoord = a\_texcoord;

}

</script>

<!-- fragment shader -->

<script id="fragment-shader-3d" type="x-shader/x-fragment">

precision mediump float;

// Passed in from the vertex shader.

varying vec2 v\_texcoord;

// The texture.

uniform sampler2D u\_texture;

void main() {

gl\_FragColor = texture2D(u\_texture, v\_texcoord);

}

</script><!--

for most samples webgl-utils only provides shader compiling/linking and

canvas resizing because why clutter the examples with code that's the same in every sample.

See https://webglfundamentals.org/webgl/lessons/webgl-boilerplate.html

and https://webglfundamentals.org/webgl/lessons/webgl-resizing-the-canvas.html

for webgl-utils, m3, m4, and webgl-lessons-ui.

-->

<script src="https://webglfundamentals.org/webgl/resources/webgl-utils.js"></script>

<script src="https://webglfundamentals.org/webgl/resources/m4.js"></script>

**CSS**@import url("https://webglfundamentals.org/webgl/resources/webgl-tutorials.css");

body {

margin: 0;

}

canvas {

width: 100vw;

height: 100vh;

display: block;

}

#ui {

width: 380px;

padding: 0.25em;

font-size: small;

}

body, document {

user-select: none;

-webkit-user-select: none;

-moz-user-select: none;

-o-user-select: none;

-ms-user-select: none;

}

#gridContainer {

position: absolute;

z-index: 2;

top: 1px;

left: 1px;

width: 400px;

height: 300px;

}

#gridContainer>svg {

width: 100%;

height: 100%;

}

#uiContainer {

left: 15px;

top: 15px;

}

#ui>div {

margin-bottom: 1em;

}

#container {

position: relative;

/\*

not sure why but on mobile webkit the gridContainer

is initially set to a large size after which

it never shrinks because it's holding the page to at

least that size.

This was one fix though not really understanding the original

issue

\*/

width: 100vw;

height: 100vh;

overflow: hidden;

}

**JS**

// WebGL - Textures - Wrap Modes

// from https://webglfundamentals.org/webgl/webgl-3d-textures-repeat-clamp.html

"use strict";

var zDepth = 50;

function main() {

// Get A WebGL context

var canvas = document.querySelector("#canvas");

var gl = canvas.getContext("webgl", {antialias: false});

if (!gl) {

return;

}

// setup GLSL program

var program = webglUtils.createProgramFromScripts(gl, ["vertex-shader-3d", "fragment-shader-3d"]);

// look up where the vertex data needs to go.

var positionLocation = gl.getAttribLocation(program, "a\_position");

var texcoordLocation = gl.getAttribLocation(program, "a\_texcoord");

// lookup uniforms

var matrixLocation = gl.getUniformLocation(program, "u\_matrix");

var textureLocation = gl.getUniformLocation(program, "u\_texture");

// Create a buffer for positions

var positionBuffer = gl.createBuffer();

// Bind it to ARRAY\_BUFFER (think of it as ARRAY\_BUFFER = positionBuffer)

gl.bindBuffer(gl.ARRAY\_BUFFER, positionBuffer);

// Put the positions in the buffer

setGeometry(gl);

// provide texture coordinates for the rectangle.

var texcoordBuffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY\_BUFFER, texcoordBuffer);

// Set Texcoords.

setTexcoords(gl);

// Create a texture.

var texture = gl.createTexture();

gl.bindTexture(gl.TEXTURE\_2D, texture);

// Fill the texture with a 1x1 blue pixel.

gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, 1, 1, 0, gl.RGBA, gl.UNSIGNED\_BYTE,

new Uint8Array([0, 0, 255, 255]));

// Asynchronously load an image

var image = new Image();

requestCORSIfNotSameOrigin(image, "https://webglfundamentals.org/webgl/resources/f-texture.png")

image.src = "https://webglfundamentals.org/webgl/resources/f-texture.png";

image.addEventListener('load', function() {

// Now that the image has loaded make copy it to the texture.

gl.bindTexture(gl.TEXTURE\_2D, texture);

gl.texImage2D(gl.TEXTURE\_2D, 0, gl.RGBA, gl.RGBA,gl.UNSIGNED\_BYTE, image);

// Check if the image is a power of 2 in both dimensions.

if (isPowerOf2(image.width) && isPowerOf2(image.height)) {

// Yes, it's a power of 2. Generate mips.

gl.generateMipmap(gl.TEXTURE\_2D);

} else {

// No, it's not a power of 2. Turn of mips and set wrapping to clamp to edge

gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_S, gl.CLAMP\_TO\_EDGE);

gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_T, gl.CLAMP\_TO\_EDGE);

gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MIN\_FILTER, gl.LINEAR);

}

drawScene();

});

var wrapS = gl.REPEAT;

var wrapT = gl.REPEAT;

document.querySelector("#wrap\_s0").addEventListener('click', function() { wrapS = gl.REPEAT; drawScene(); }); // eslint-disable-line

document.querySelector("#wrap\_s1").addEventListener('click', function() { wrapS = gl.CLAMP\_TO\_EDGE; drawScene(); }); // eslint-disable-line

document.querySelector("#wrap\_s2").addEventListener('click', function() { wrapS = gl.MIRRORED\_REPEAT; drawScene(); }); // eslint-disable-line

document.querySelector("#wrap\_t0").addEventListener('click', function() { wrapT = gl.REPEAT; drawScene(); }); // eslint-disable-line

document.querySelector("#wrap\_t1").addEventListener('click', function() { wrapT = gl.CLAMP\_TO\_EDGE; drawScene(); }); // eslint-disable-line

document.querySelector("#wrap\_t2").addEventListener('click', function() { wrapT = gl.MIRRORED\_REPEAT; drawScene(); }); // eslint-disable-line

function isPowerOf2(value) {

return (value & (value - 1)) === 0;

}

function radToDeg(r) {

return r \* 180 / Math.PI;

}

function degToRad(d) {

return d \* Math.PI / 180;

}

var fieldOfViewRadians = degToRad(60);

drawScene();

window.addEventListener('resize', drawScene);

// Draw the scene.

function drawScene() {

webglUtils.resizeCanvasToDisplaySize(gl.canvas);

// Tell WebGL how to convert from clip space to pixels

gl.viewport(0, 0, gl.canvas.width, gl.canvas.height);

// Clear the framebuffer texture.

gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

gl.enable(gl.CULL\_FACE);

gl.enable(gl.DEPTH\_TEST);

// Compute the matrix

var scaleFactor = 2.5;

var tsize = 80 \* scaleFactor;

var x = gl.canvas.clientWidth / 2 - tsize / 2;

var y = gl.canvas.clientHeight - tsize - 60;

gridContainer.style.left = (x - 50 \* scaleFactor) + 'px';

gridContainer.style.top = (y - 50 \* scaleFactor) + 'px';

gridContainer.style.width = (scaleFactor \* 400) + 'px';

gridContainer.style.height = (scaleFactor \* 300) + 'px';

// Tell it to use our program (pair of shaders)

gl.useProgram(program);

// Turn on the position attribute

gl.enableVertexAttribArray(positionLocation);

// Bind the position buffer.

gl.bindBuffer(gl.ARRAY\_BUFFER, positionBuffer);

// Tell the position attribute how to get data out of positionBuffer (ARRAY\_BUFFER)

var size = 3; // 3 components per iteration

var type = gl.FLOAT; // the data is 32bit floats

var normalize = false; // don't normalize the data

var stride = 0; // 0 = move forward size \* sizeof(type) each iteration to get the next position

var offset = 0; // start at the beginning of the buffer

gl.vertexAttribPointer(

positionLocation, size, type, normalize, stride, offset);

// Turn on the texcoord attribute

gl.enableVertexAttribArray(texcoordLocation);

// bind the texcoord buffer.

gl.bindBuffer(gl.ARRAY\_BUFFER, texcoordBuffer);

// Tell the texcoord attribute how to get data out of texcoordBuffer (ARRAY\_BUFFER)

var size = 2; // 2 components per iteration

var type = gl.FLOAT; // the data is 32bit floats

var normalize = false; // don't normalize the data

var stride = 0; // 0 = move forward size \* sizeof(type) each iteration to get the next position

var offset = 0; // start at the beginning of the buffer

gl.vertexAttribPointer(

texcoordLocation, size, type, normalize, stride, offset);

// Compute the projection matrix

var projectionMatrix =

m4.orthographic(0, gl.canvas.clientWidth, gl.canvas.clientHeight, 0, -1, 1);

gl.bindTexture(gl.TEXTURE\_2D, texture);

gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MIN\_FILTER, gl.NEAREST);

gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_S, wrapS);

gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_WRAP\_T, wrapT);

var matrix = m4.translate(projectionMatrix, x, y, 0);

matrix = m4.scale(matrix, tsize, tsize, 1);

matrix = m4.translate(matrix, 0.5, 0.5, 0);

// Set the matrix.

gl.uniformMatrix4fv(matrixLocation, false, matrix);

// Tell the shader to use texture unit 0 for u\_texture

gl.uniform1i(textureLocation, 0);

// Draw the geometry.

gl.drawArrays(gl.TRIANGLES, 0, 1 \* 6);

}

}

// Fill the buffer with the values that define a plane.

function setGeometry(gl) {

var positions = new Float32Array(

[

-0.5, 0.5, 0.5,

0.5, 0.5, 0.5,

-0.5, -0.5, 0.5,

-0.5, -0.5, 0.5,

0.5, 0.5, 0.5,

0.5, -0.5, 0.5,

]);

gl.bufferData(gl.ARRAY\_BUFFER, positions, gl.STATIC\_DRAW);

}

// Fill the buffer with texture coordinates for a plane.

function setTexcoords(gl) {

gl.bufferData(

gl.ARRAY\_BUFFER,

new Float32Array(

[

-3, -1,

2, -1,

-3, 4,

-3, 4,

2, -1,

2, 4,

]),

gl.STATIC\_DRAW);

}

main();

// This is needed if the images are not on the same domain

// NOTE: The server providing the images must give CORS permissions

// in order to be able to use the image with WebGL. Most sites

// do NOT give permission.

// See: https://webglfundamentals.org/webgl/lessons/webgl-cors-permission.html

function requestCORSIfNotSameOrigin(img, url) {

if ((new URL(url, window.location.href)).origin !== window.location.origin) {

img.crossOrigin = "";

}

}